Compulsory 2

Sigbjørn Fjelland

2022-05-06

Task 1

Problem 1.1

```
library(bootstrap)
library(boot)
library(ggplot2)
library(data.table)
library(Rcpp)
library(VGAM)
```

```
## Loading required package: stats4
```

```
## Loading required package: splines
```

```
##
## Attaching package: 'VGAM'
```

```
## The following objects are masked from 'package:boot':
##
##     logit, simplex
```

```
## The following object is masked from 'package:bootstrap':
##
##     hormone
```

```
library(MASS)
```

```
# Presenting the data
data <- scor
data.open <- c("mec", "vec")
data.closed <- c("alg", "ana", "sta")
all.sets <- c(data.open, data.closed)

#constructing groups that are to be correlated as functon for the boot package

correlating.groups <- function(data, ind){
                    mek <- data[ind, 1]
                    vec <- data[ind, 2]
                    alg <- data[ind, 3]
                    ana <- data[ind, 4]
                    sta <- data[ind, 5]


        c(cor(mek,vec), cor(alg,ana), cor(alg, sta), cor(ana,sta))
```

```
}
```

```
boot(data = data, statistic = correlating.groups, R = 200)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = data, statistic = correlating.groups, R = 200)
##
##
## Bootstrap Statistics :
##      original        bias    std. error
## t1* 0.5534052  0.0015970223  0.07349188
## t2* 0.7108059  0.0032297071  0.05060925
## t3* 0.6647357 -0.0008492849  0.05792702
## t4* 0.6071743  0.0096818818  0.06458413
```

problem 1.2

```
theta.hat <- function(data, ind){
                    mek <- data[ind, 1]
                    vec <- data[ind, 2]
                    alg <- data[ind, 3]
                    ana <- data[ind, 4]
                    sta <- data[ind, 5]

                    data <- cbind(mek, vec, alg, ana, sta)
                    lambda <- eigen(cov(data))$values

                    lambda[1]/sum(lambda)
                    }
```

```
theta.boot <- boot(data = data, statistic = theta.hat, R=200, mle = eigen(cov(data)))
theta.boot
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = data, statistic = theta.hat, R = 200, mle = eigen(cov(data)))
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1* 0.619115 0.004075202   0.0474656
```

problem 1.3

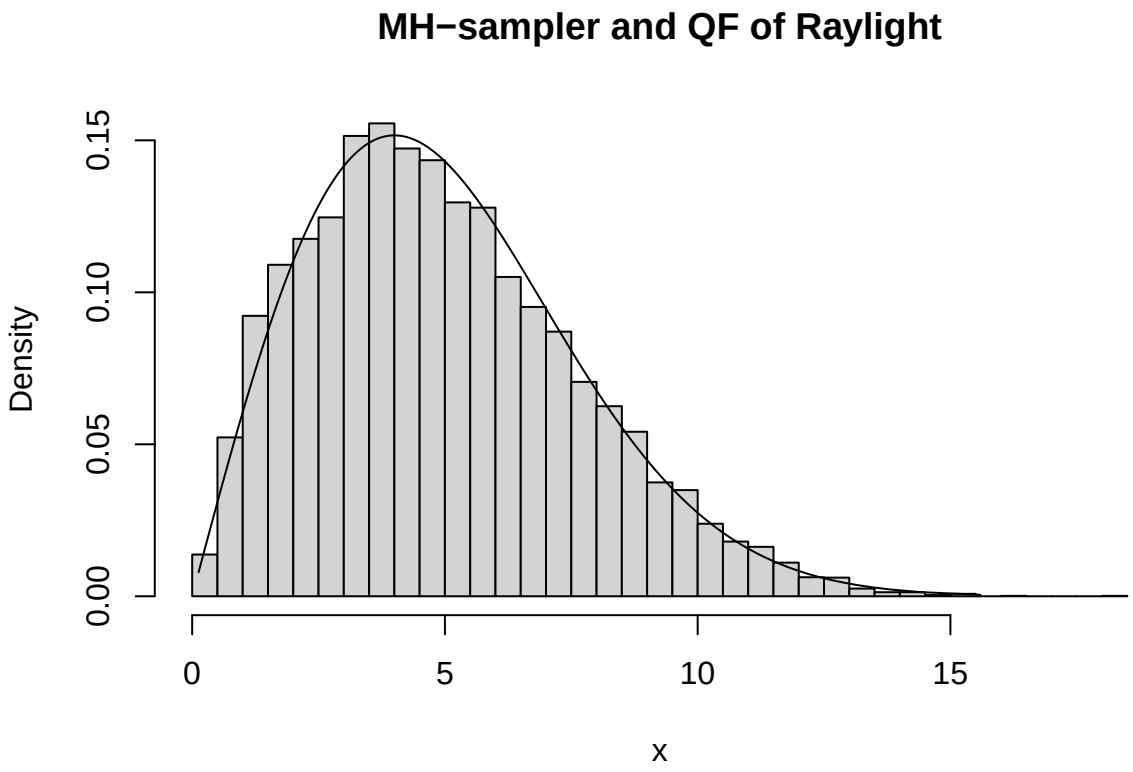
```
print(boot.ci(theta.boot,type = "perc"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 200 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = theta.boot, type = "perc")
##
## Intervals :
## Level     Percentile
## 95%   ( 0.5181,  0.7093 )
## Calculations and Intervals on Original Scale
## Some percentile intervals may be unstable
```

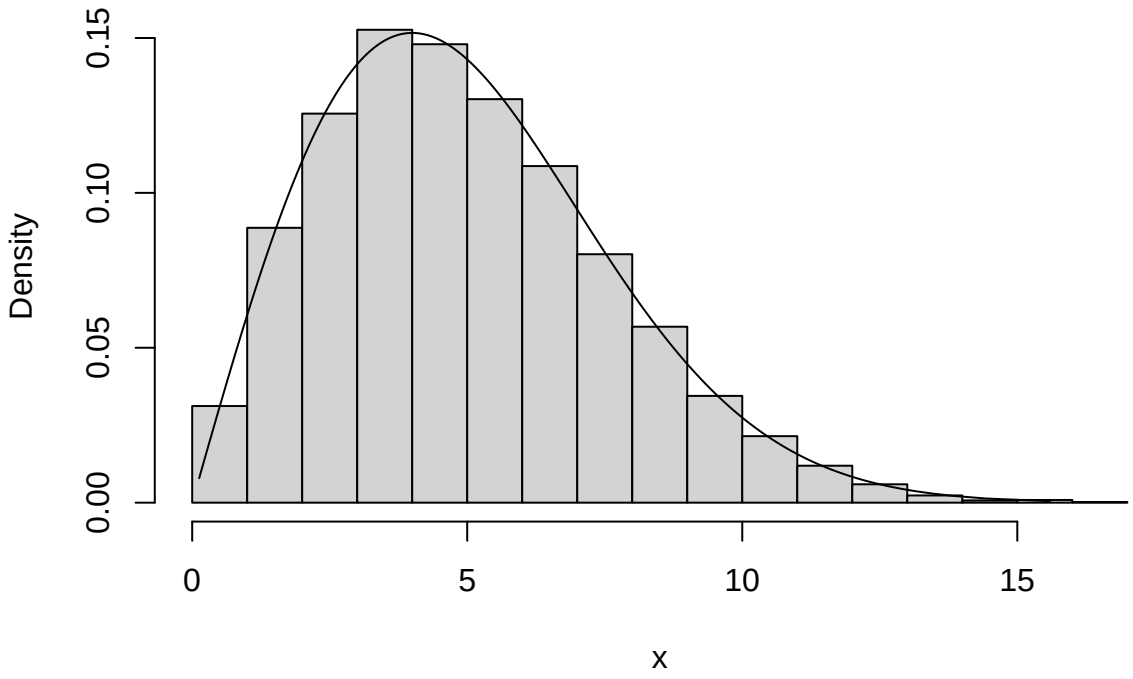
Task 2

problem 2.1 MH-Sampler with rayleigh distribution as proposal distribution

```
set.seed(123)

#Proposal distribution (Rayleigh)
f <- function(x, sigma) {
  if (any(x < 0)) return (0)
  stopifnot(sigma > 0)
  return((x / sigma^2) * exp(-x^2 / (2*sigma^2)))
}


Nsim <- 20000 #total simulations
burnin = 5000 #sample to burn
sigma <- 4


x <- numeric(Nsim)
x[1] <- rgamma(1, rate = 1, shape = 1) # start value to simplify index in loop
k <- 0 ## conter for rejects ##
u <- runif(Nsim)

for (i in 2:Nsim) {
  xt <- x[i-1]
  y <- rgamma(1, shape = xt, rate = 1)
  num <- f(y, sigma) * dgamma(xt, rate = 1, shape = y)  #the numerator of r(xt,y)
  den <- f(xt, sigma) * dgamma(y, rate = 1, shape = xt) #the denominator of r(xt,y)
  if (u[i] <= num/den) x[i] <- y else {
    x[i] <- xt
    k <- k+1 # Number of y being rejected
  }
}

index<-seq(10,Nsim,by=10)  ## take index every 10 steps

#sampled distribution from MH (and library for compartion)
sampled_ray_dist <- x[burnin:Nsim]
raylib <- rrayleigh(Nsim+1-burnin, scale = sigma)


#quantile function
a <- ppoints(1000)
QR <- sigma * sqrt(-2 * log(1 - a))

#Hists
hist(sampled_ray_dist, breaks="scott",
     main="MH-sampler and QF of Raylight", xlab="x", freq=FALSE)
lines(QR, f(QR, 4))
```




```
hist (raylib,freq=FALSE,
      main = "Rayl. dist. from lib. sampler and QF.", xlab = "x")
lines(QR, f(QR, 4))
```



Problem 2.2

Vi Benytter normal fordelingen som symetrisk distribusjon, med $\sigma = (0.05, 0.5, 2, 16)$, og på gøy lager vi en sidekick med laplace

```
rw.Metropolis.normal <- function(n, sigma, x0, N) {
        x <- numeric(N)
        x[1] <- x0
        u <- runif(N)
        k <- 0
          for (i in 2:N) {
              y <- rnorm(1, x[i-1], sigma) #rlaplace(n=1, location=x[i-1], scale = sigma) #
                  if (u[i] <= (dt(y, n) / dt(x[i-1], n)))
                  x[i] <- y else {
                  x[i] <- x[i-1]
                      k <- k + 1
                      }
              }
        return(list(x=x, k=k))
}



rw.Metropolis.laplace <- function(n, sigma, x0, N) {
        x <- numeric(N)
        x[1] <- x0
        u <- runif(N)
        k <- 0
          for (i in 2:N) {
              y <- rlaplace(n=1, location=x[i-1], scale = sigma) #
                  if (u[i] <= (dt(y, n) / dt(x[i-1], n)))
                  x[i] <- y else {
                  x[i] <- x[i-1]
                      k <- k + 1
                      }
              }
        return(list(x=x, k=k))
}
```

Random Walk for samplet fra normal fordelingen:

```
n <- 4 #degrees of freedom for target Student t dist.
N <- 2000
sigma <- c(.05, .5, 2, 16)
x0 <- 25
rw1.n <- rw.Metropolis.normal(n, sigma[1], x0, N)
rw2.n <- rw.Metropolis.normal(n, sigma[2], x0, N)
rw3.n <- rw.Metropolis.normal(n, sigma[3], x0, N)
rw4.n <- rw.Metropolis.normal(n, sigma[4], x0, N)

# Result from the random walks given sigma vector
print(c(rw1.n$k, rw2.n$k, rw3.n$k, rw4.n$k))
```

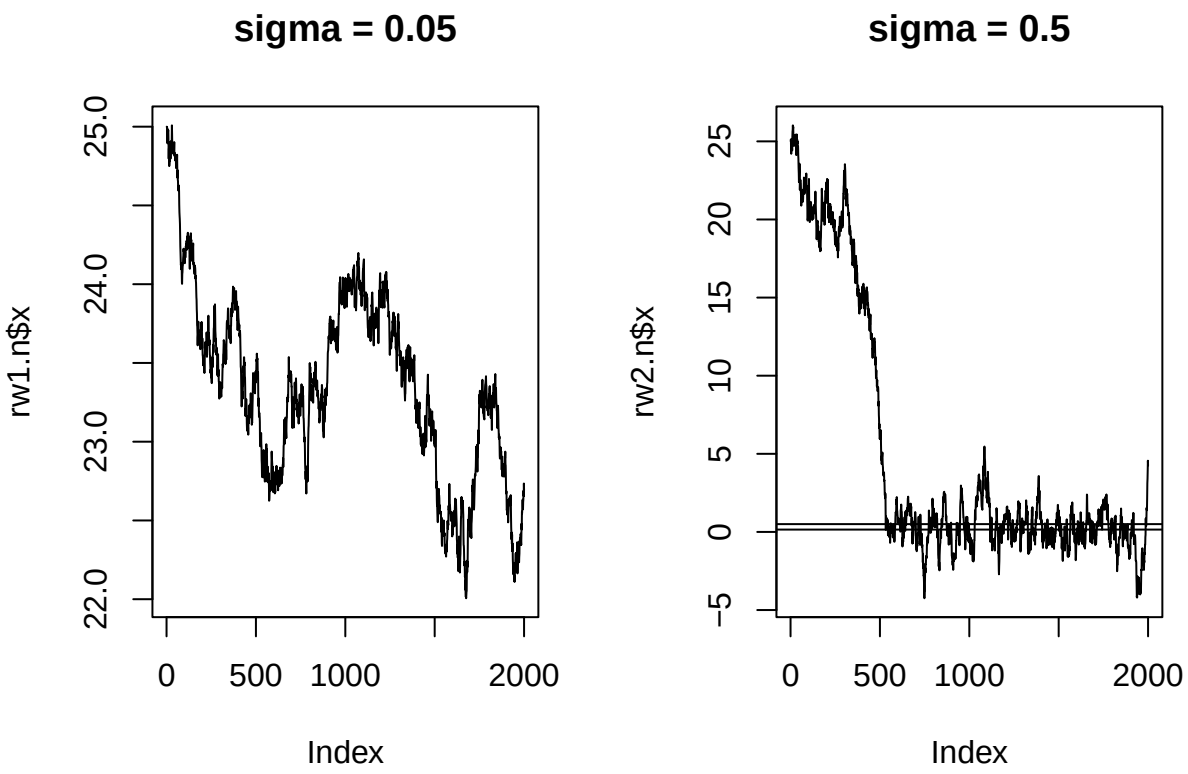
```
## [1]    7  225  911 1804
```

```
#Proportion of rejections within [0.15, 0.5]
print(c((2000 - length(which(rw1.n$x > 0.5)) - length(which(rw1.n$x < 0.15)))/N,
        (2000 - length(which(rw2.n$x > 0.5)) - length(which(rw2.n$x < 0.15)))/N,
        (2000 - length(which(rw3.n$x > 0.5)) - length(which(rw3.n$x < 0.15)))/N,
        (2000 - length(which(rw4.n$x > 0.5)) - length(which(rw4.n$x < 0.15)))/N))
```

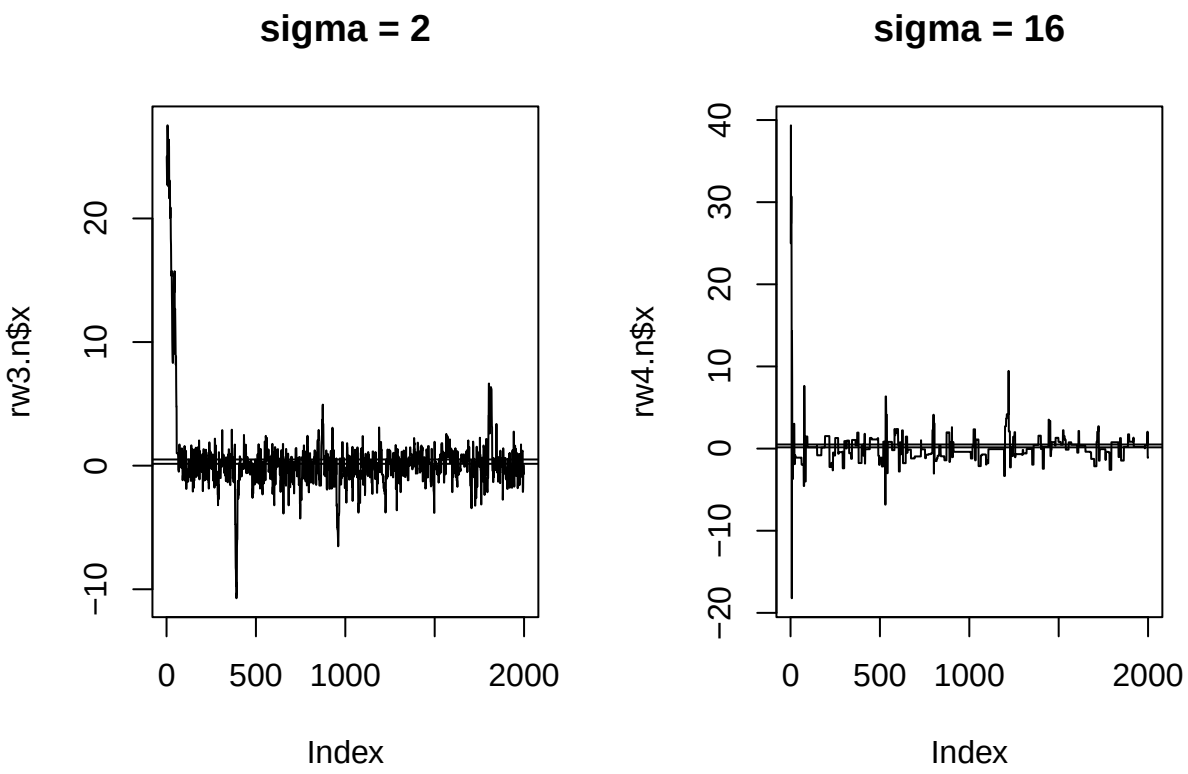
```
## [1] 0.000 0.099 0.116 0.167
```

Vi ser at både $\sigma = 2$ og $\sigma = 16$ har en lik andelen innenfor intervallet. For å få en illustrasjon på hvilke σ som konvergerer plotter vi resulktatet:

```
#Plot of sigma1 and sigma2
par(mfrow=c(1,2))

plot(rw1.n$x, type = "l", main="sigma = 0.05")
abline(h=c(0.15,0.5))

plot(rw2.n$x, type = "l",main="sigma = 0.5")
abline(h=c(0.15,0.5))
```




```
#Plot of sigma1 and sigma2
par(mfrow=c(1,2))
plot(rw3.n$x, type = "l",main="sigma = 2")
abline(h=c(0.15,0.5))

plot(rw4.n$x, type = "l",main="sigma = 16")
abline(h=c(0.15,0.5))
```



En ser på plottet at både $\sigma = 0.5$, $\sigma = 2$ og $\sigma = 16$ konvergerer innenfor intervallet, men $\sigma = 0.5$ trenger en lengre periode med burning før den setler seg i en omegn av intervallet. $\sigma = 16$ ser noe mindre effektiv ut. Foretrukken σ å optimalisere rundt ville vert $\sigma = 2$.

og så til vår lille sidekick:

```
n <- 4 #degrees of freedom for target Student t dist.
N <- 2000
sigma <- c(.05, .5, 2, 16)
x0 <- 25
rw1.l <- rw.Metropolis.laplace(n, sigma[1], x0, N)
rw2.l <- rw.Metropolis.laplace(n, sigma[2], x0, N)
rw3.l <- rw.Metropolis.laplace(n, sigma[3], x0, N)
rw4.l <- rw.Metropolis.laplace(n, sigma[4], x0, N)

# Result from the random walks given sigma vector
print(c(rw1.l$k, rw2.l$k, rw3.l$k, rw4.l$k))
```

```
## [1]    9  327  953 1751
```

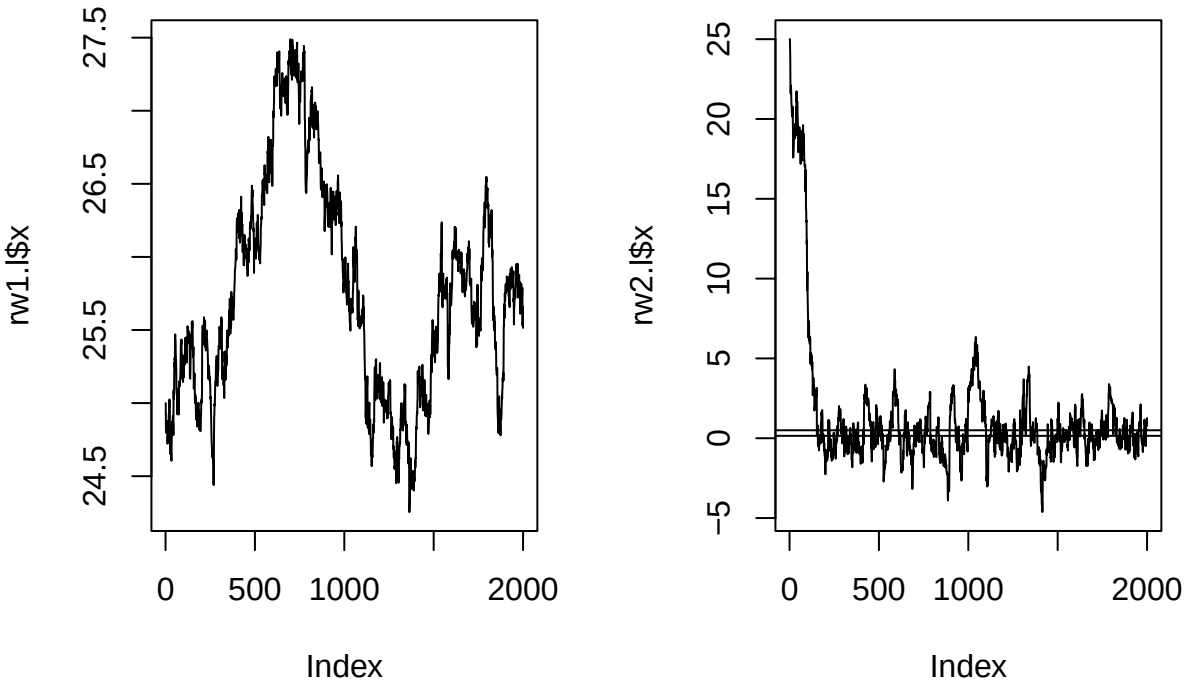
```
#Proportion of rejections within [0.15, 0.5]
print(c((2000 - length(which(rw1.l$x > 0.5)) - length(which(rw1.l$x < 0.15)))/N,
        (2000 - length(which(rw2.l$x > 0.5)) - length(which(rw2.l$x < 0.15)))/N,
        (2000 - length(which(rw3.l$x > 0.5)) - length(which(rw3.l$x < 0.15)))/N,
        (2000 - length(which(rw4.l$x > 0.5)) - length(which(rw4.l$x < 0.15)))/N))
```

```
## [1] 0.000 0.108 0.119 0.216
```

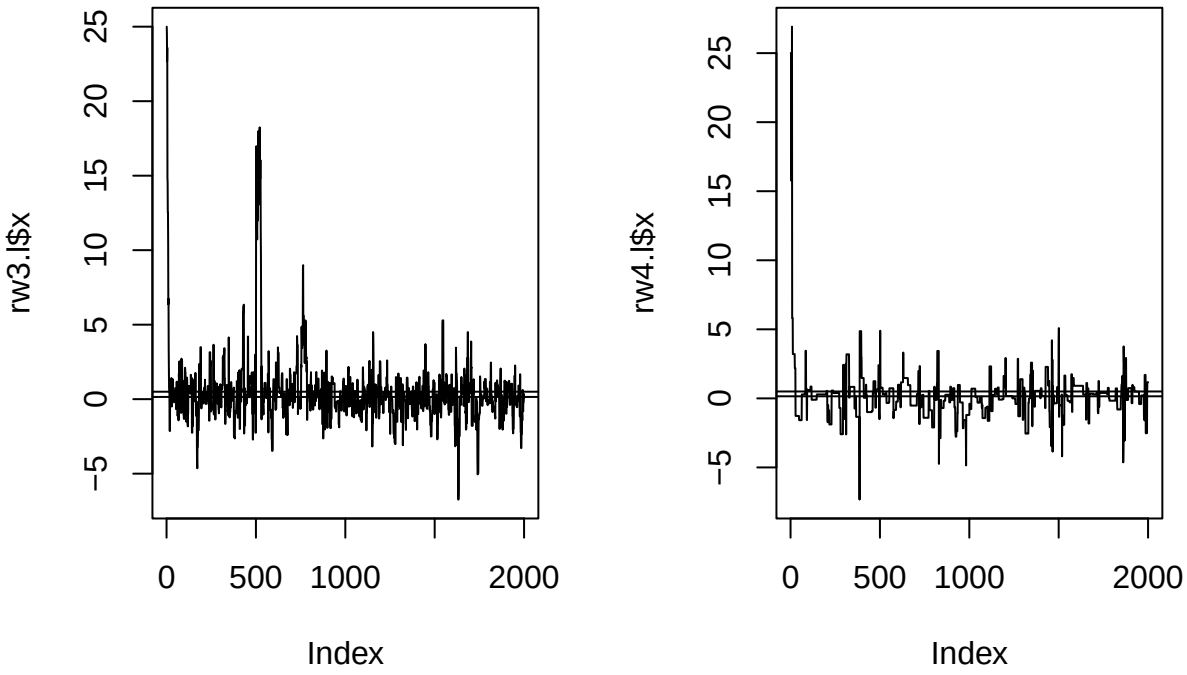
For å se for hvilke σ som konvergerer

```
#Plot of sigma1 and sigma2
par(mfrow=c(1,2))
```

```
plot(rw1.l$x, type = "l")
abline(h=c(0.15,0.5))

plot(rw2.l$x, type = "l")
abline(h=c(0.15,0.5))
```



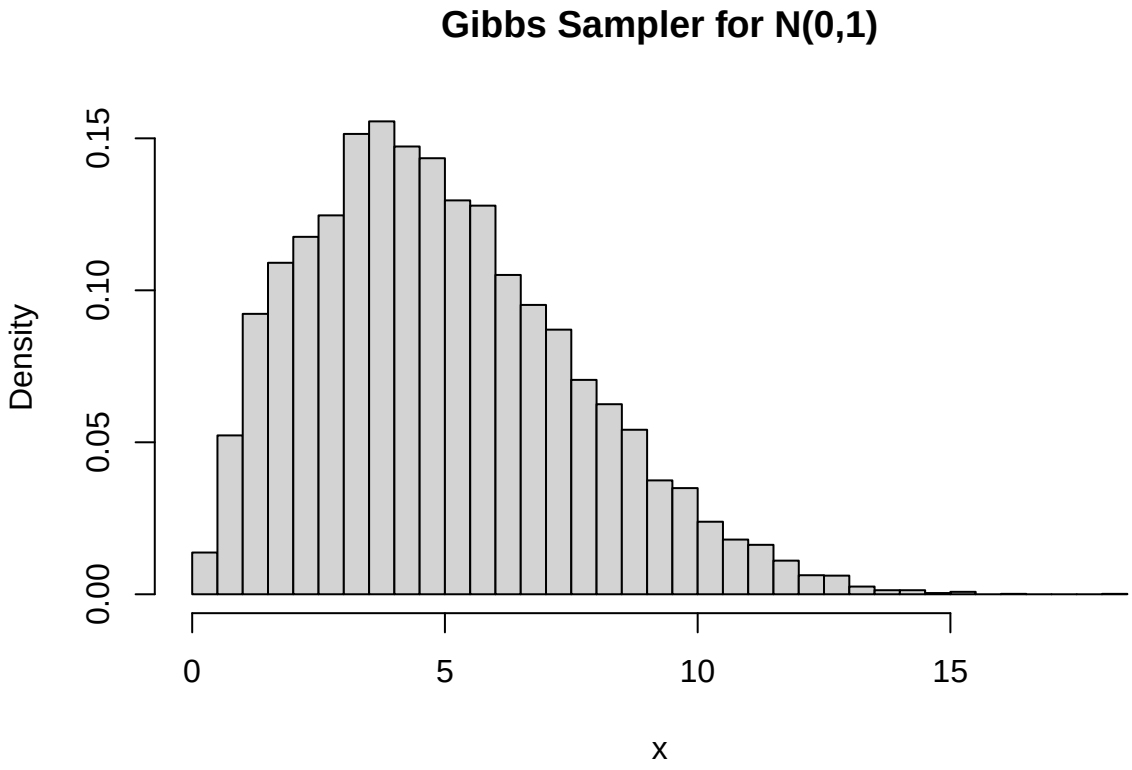

```
#Plot of sigma1 and sigma2
par(mfrow=c(1,2))
plot(rw3.l$x, type = "l")
abline(h=c(0.15,0.5))

plot(rw4.l$x, type = "l")
abline(h=c(0.15,0.5))
```



Det må sies å være til forveksling likt i konvergens i tillegg konvergerer $\sigma = 0.05$ noe fortere.

Problem 2.3

```
set.seed(123)

#initialize constants and parameters
N <- 5000 #length of chain
burn <- 1000 #burn-in length
X <- matrix(0, N, 2) #the chain, a bivariate sample
rho <- -.9 #correlation
mu_x <- 0
mu_y <- 0
sigma_x <- 1
sigma_y <- 1
s1 <- sqrt(1-rho^2)*sigma_x
s2 <- sqrt(1-rho^2)*sigma_y


###### generate the chain #####
X[1, ] <- c(mu_x, mu_y) #initialize
for (i in 2:N) {
x2 <- X[i-1, 2]
m_x <- mu_x + rho * (y - mu_y) * sigma_x/sigma_y
X[i, 1] <- rnorm(1, m_x, s1)
x_x <- X[i, 1]
m_y <- mu_y + rho * (x_x - mu_x) * sigma_y/sigma_x
X[i, 2] <- rnorm(1, m_y, s2)
}
b <- burn + 1
x <- X[b:N, ]

hist(sampled_ray_dist, breaks="scott",
     main="Gibbs Sampler for N(0,1)", xlab="x", freq=FALSE)
```


Gibbs Sampler for N(0,1)
Density
0.00
0.05
0.10
0.15
0
5
10
15
x

Task 3

problem 3.1

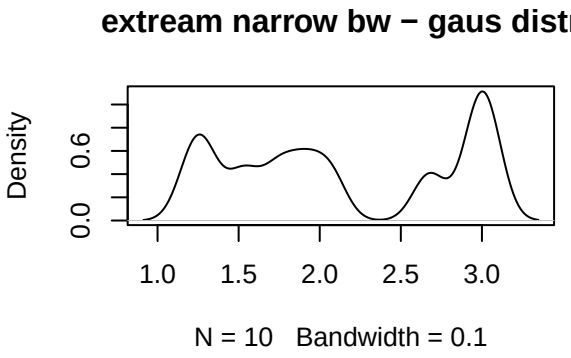
```
p1 <- 0.5
n = 10
data <- p1 *rnorm(n, 1,1) + (1-p1) * rnorm(n, 3,1)
data1 <- p1 *rgamma(n, rate = 1, shape = 1) + (1-p1) * rgamma(n, rate = 3, shape = 1)
par(mfrow = c(1, 1))
print(density(data))
```

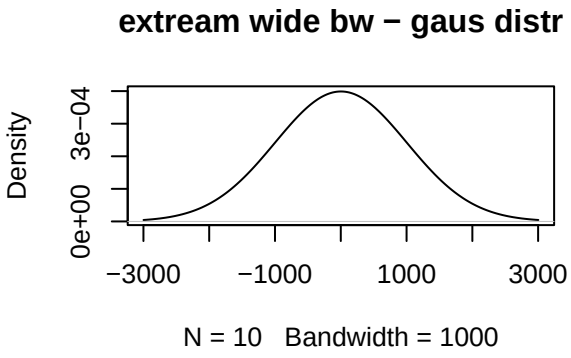
```
##
## Call:
##  density.default(x = data)
##
## Data: data (10 obs.);    Bandwidth 'bw' = 0.4101
##
##        x                 y
##  Min.   :-0.01707   Min.   :0.001771
##  1st Qu.: 1.05666   1st Qu.:0.053027
##  Median : 2.13038   Median :0.283447
##  Mean   : 2.13038   Mean   :0.232491
##  3rd Qu.: 3.20411   3rd Qu.:0.374453
##  Max.   : 4.27783   Max.   :0.455501
```

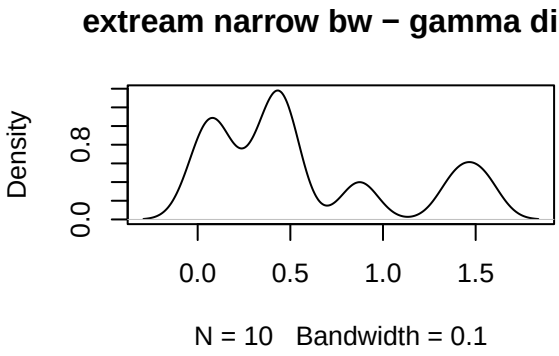
```
df <- approxfun(density(data))   ##extract the density function##
xnew <- seq(0,10,0.2)
n <- length(data)
h1 <- 0.9 * min(c(IQR(data)/1.34, sd(data))) * n^(-1/5) #density seems not unimodal#
h2 <- 1.06 * sd(data) * n^(-1/5)
```

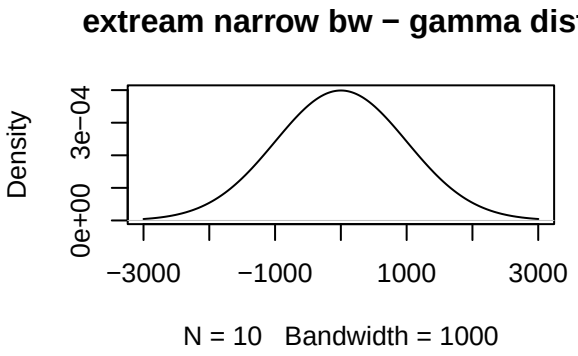
lets first look at the extream example (high, low):

```
par(mfrow = c(2, 2))
plot(density(data,bw=.10) , main="extream narrow bw - gaus distr") # "Day of the tentacle"
plot(density(data,bw=1000), main="extream wide bw - gaus distr")  #Glææt - "too much
plot(density(data1,bw=.10), main="extream narrow bw - gamma distr") # "Day of the tentacle"
plot(density(data1,bw=1000), main="extream narrow bw - gamma distr")  #Glææt - "too much
```



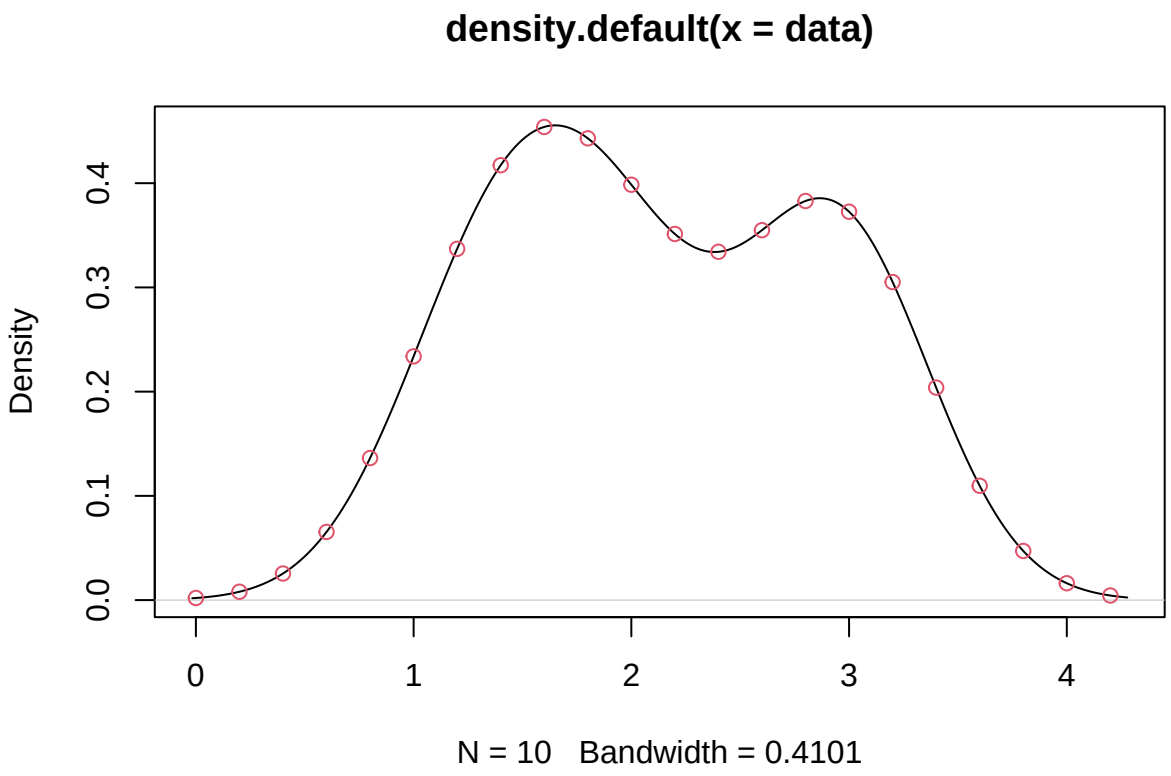




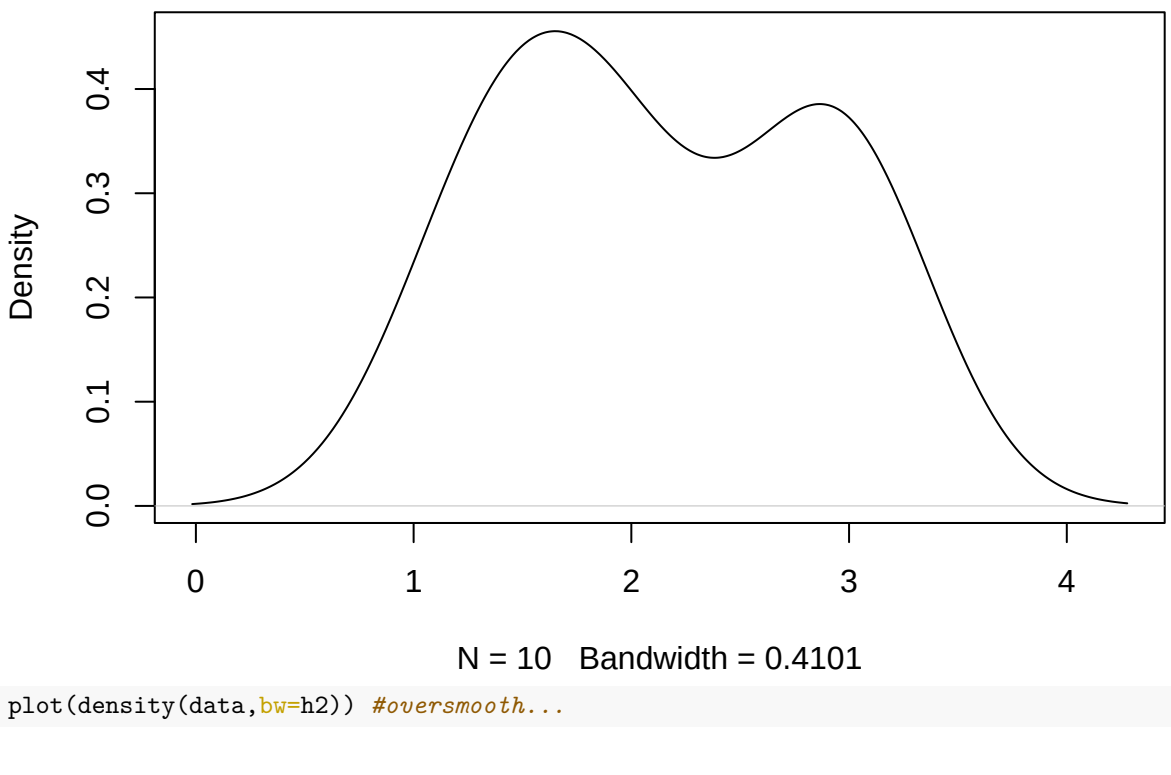


From this we can conclude that choise of bandwith is essential. To narrow you get spikes, to wide you obtain a verry flat, smooth curve converged towards the normal distribution. As we also see the result is the same if we change the kernel distribution. So lets try to fit a better curve:

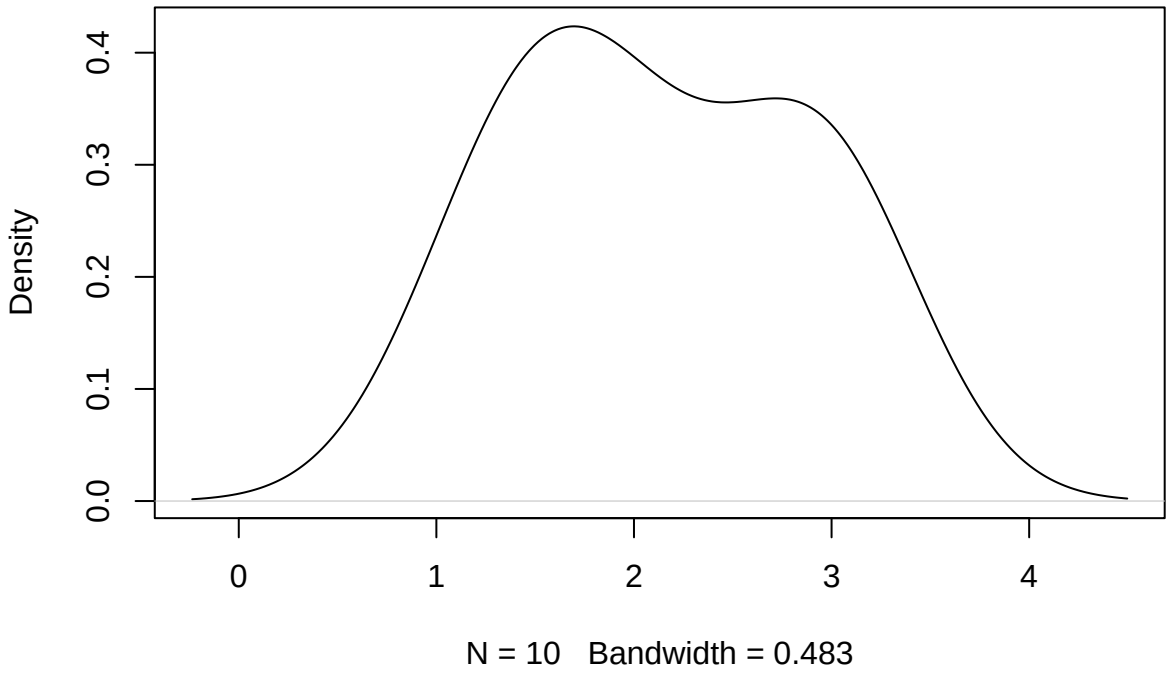
```
#par(mfrow=c(2,2))
plot(density(data)) #The default method applies the Gaussian kernel#3
points(xnew,df(xnew),col=2)  #bulky
```




```
plot(density(data,bw=h1)) #Better, this include the duality of the two distr.
```




```
plot(density(data,bw=h2)) #oversmooth...
```


density.default(x = data, bw = h2)
Density
0.0
0.1
0.2
0.3
0.4
0
1
2
3
4
N = 10 Bandwidth = 0.483

problem 3.2

we have:

$$\begin{aligned}\mathbf{x} =&\{x_1, ..., x_n\} \sim gamma(\theta)\\ \theta =&(r, \lambda)\end{aligned}$$

we first need to solve for the MLE functions for the gamma distributed RVs:

$$\begin{aligned}f(\mathbf{x}|\theta) =&\frac{\lambda^r}{\Gamma(r)} \cdot \mathbf{x}^{r-1} \cdot exp(-\lambda \mathbf{x})\\ L(\theta|x_1, ..., x_n) =&\prod_{i=1}^{n} f(\mathbf{x}|\theta)\\ =&\frac{\lambda^{n \cdot r}}{\Gamma(r)^n} \cdot \sum_i^n x_i^{r-1} \cdot exp(-\lambda \sum_i^n x_i)\end{aligned}$$

we obtain the log likeliehood by applying magic

$$l(\theta|x_1, ..., x_n) =nr \cdot log(\lambda) - n \cdot log(\Gamma(r)) + (r-1) \cdot \sum_i^n log(x_i) - \lambda \sum_i^n x_i$$

we then take the derivative to find the maximum:

$$\begin{aligned}\frac{\partial l}{\partial \lambda} =&\frac{nr}{\lambda} - \sum_{i=1}^{n} x_i = 0\\ \frac{\partial l}{\partial r} =&n \cdot log(\lambda) - \frac{n\Gamma'(r)}{\Gamma(r)} + \sum_{i=1}^{n} log(x_i) = 0\end{aligned}$$

to find $\hat{\lambda}$ is fairly straight forward:

$$\frac{nr}{\lambda} - \sum_{i=1}^{n} x_i = 0$$
$$\frac{nr}{\lambda} = \sum_{i=1}^{n} x_i$$
$$\hat{\lambda} = \frac{nr}{\sum_{i=1}^{n} x_i} = \frac{r}{n^{-1}\sum_{i=1}^{n} x_i} = \frac{\hat{r}}{\bar{x_n}}$$

to find $\hat{r}$ is more of a pain since we have the $\frac{\Gamma'(r)}{\Gamma(r)}$ term also known as digamma:

$$\psi(r) = \frac{\Gamma'(r)}{\Gamma(r)} = \frac{\partial}{\partial r} log(\Gamma(r))$$

to simplify we substitute $\hat{\lambda}$ into the equation:

$$n \cdot log(\frac{\hat{r}}{\bar{x_n}}) - \frac{n\Gamma'(\hat{r})}{\Gamma(\hat{r})} + \sum_{i=1}^{n} log(x_i) = 0 \tag{1}$$

$$log(\frac{\hat{r}}{\bar{x_n}}) - \frac{1}{n}\sum_{i=1}^{n} log(x_i) = \frac{\Gamma'(\hat{r})}{\Gamma(\hat{r})} = \psi(\lambda \bar{x_n}) \tag{2}$$

for $\bar{x_n} = \frac{r}{\lambda}$

Due to the digamma function the MLE needs to be solved numericaly by root solver (in r uniroots), which is presumably a bisection method, since derivative is not needed. Bisection only need a continous function to converge to one of the existing roots. Bellow the same simulation asfrom "Statistical computing with R by Maria L. Rizzo" page 340 is conducted (with comments on the process)

```
m <- 20000
est <- matrix(0, m, 2)
n <- 200
r <- 5
lambda <- 2

# equation (1) above, of which we want to find the root
obj <- function(lambda, xbar, logx.bar) {
    digamma(lambda * xbar) - logx.bar - log(lambda)
    }

#Do m simulations of a set of x_1,...,x_n
for (i in 1:m) {
    x <- rgamma(n, shape=r, rate=lambda)
    xbar <- mean(x)

    # using uniroot to solve numericaly
    u <- uniroot(obj, lower = .001, upper = 10e5,
        xbar = xbar, logx.bar = mean(log(x)))

  #columns vector of MLE simulations
  lambda.hat <- u$root
  r.hat <- xbar * lambda.hat
  est[i, ] <- c(r.hat, lambda.hat)
}

#mean of the MLE simulations

ML <- colMeans(est)

ML
```

```
## [1] 5.070865 2.030602
```